Appendices A–D
A Scalable Method for Deductive Generalization in the Spreadsheet Paradigm

MARGARET BURNETT
Oregon State University
SHERRY YANG
Oregon Institute of Technology
and
JAY SUMMET
Oregon State University

APPENDIX A. THE CG EDGE MAINTENANCE ALGORITHMS

CG edge maintenance is done using the simple algorithms `Insert`, `Delete`, and `Find`, shown in Figure A-1.

```
//Add labeledEdge le = (label,(u,v))
Algorithm Insert(le)
  foundLE = Find(le);
  if foundLE = notFound
     then add le to le.v.edges
     else foundLE.counter = foundLE.counter + 1;
```

```
//Delete labeledEdge le = (label(e),(u,v))
Algorithm Delete(le)
  if le's location is known
     then foundLE = le
     else foundLE = Find(le);
  if foundLE.counter = 1
     then remove foundLE from foundLE.v.edges
     else foundLE.counter = foundLE.counter - 1;
```

```
//Find labeledEdge le = (label(e),(u,v))
Algorithm Find(le)
  let v = le.v    //v is the cell where le should be stored.
  if le is in v.edges    then return le
                         else return notFound;
```

Fig. A-1. Basic algorithms for edge maintenance.

APPENDIX B. COMPUTATIONAL EQUIVALENCE BETWEEN CONCRETE AND GENERALIZED FORMULAS

We will say that the generalized version of a reference in cell X's formula is *computationally equivalent* to the concrete version if replacing every concrete reference $F_i : Z$ with the generalized reference results in the same value in cell X as with the concrete reference, provided that the concrete version terminates. In this appendix, we show that the generalization method maintains this property. The notion of computational equivalence is basically correctness of the method under the modelessness constraint given in the introduction, which requires that there cannot be a "pregeneralization mode" that requires user actions or reasoning that are different from those of a "postgeneralization mode."

For now, assume the cell reference graph has information about *all* relationships in the program, i.e., that there are no off-screen cells, and that no ICE edges are present in the concrete version. (We will relax these assumptions in the next paragraph.) Under these assumptions, as we have pointed out before, replacing every reference $F_i : Z$ with $F(\mathit{DefSet}_i) : Z$ using Notation 1 would have maintained this property, since $F(\mathit{DefSet}_i)$ completely describes F_i by enumerating every difference between F_i and F. Despite its omission of information from Notation 1, Notation 2 does not lose this property, because a cell not in $\mathit{AffectsSet}_x$ cannot possibly have any effect on X's value; hence a reference to $F_i : Z = F(\mathit{DefSet}_i \cap \mathit{AffectsSet}_x) : Z$ must produce the same result in X as a reference to $F(\mathit{DefSet}_i){:}Z$ would produce. Hence the generalized version of the formula under both Notations 1 and 2 are computationally equivalent to the concrete version under these assumptions, provided that the concrete version terminates.

We now relax the above assumptions. First, we will allow some of the relationships in the program to be off-screen (and thus not in the cell reference graph). Because a trigger for generalizing is removal of a cell from the screen, we know that all relationships involving off-screen cells were generalized at the time the cell was removed. We have already explained that it is neither possible to change the relationships uniquely described in existing off-screen cells, nor to add new references to these off-screen cells without bringing them back onto the screen. Because of this property, all relationships involving an off-screen cell have already been generalized in a way that cannot be affected by the user's manipulations and edits of on-screen information, and the cell reference graph can safely omit information about them. Even when off-screen cell formulas are generated via gestures, the generated formulas are created in pregeneralized form; after their creation the only way they can be edited in a way that requires new generalization activity is for them to be brought onto the screen.

Second, we will allow ICE edges to be present in the concrete version. Although these edges are removed from the cell reference graph, this does not result in loss of information, because the inherited references on form instances that these edges model are, by definition, simply duplicates of references that also exist in the model. Thus, the model form's edges include the same information as the ICE edges and it is not necessary to include the latter's duplicated information in order to generalize. It is also not necessary to actually generalize the inherited formulas modeled by the ICE edges, because cells with "inherited formulas" can just copy or use the formulas in the model version after generalization is complete.

Thus, the method satisfies the computational equivalence property for concrete forms that are able to terminate. We now consider concrete forms that do not terminate.

Concrete forms that do not terminate, due to possible cycles, are either incorrectly introducing true cycles, or are attempting to introduce recursion. Formulas introducing true cycles are rejected, so no further attention to this situation is needed. For concrete forms attempting to introduce recursion, we now show that the "correct" recursive form is deduced, by induction on the number of recursive instances (references to other instances of the same form). By "correct," we mean that the recursive form has the same relationships, except for the ICE edges, as those of the concrete forms.

The base case of the recursion (and also the base case of our induction)—references that do not refer to other instances of the same form—has already been demonstrated above. The inductive hypothesis is that $n-1$ recursive instances are correct. From this, it is obvious that the nth recursive instance is itself correct because it combines these instances using exactly the same operators as those specified concretely.

Finally, we show that generalization does not introduce infinite recursion if the concrete version did not do so. The key to this lies in the *AffectsSet*: the system does not include any cells in the generalized notation except those on the direct dataflow path to the formula being generalized, in essence following a lazy strategy. More generally, the system does not add any relationships that were not already specified by the user, and in fact eliminates all except those directly needed to compute the cell being generalized. Thus, since it does not add new relationships and eliminates those not needed for computation of the formula's result, it cannot introduce infinite recursion if the concrete version did not do so.

APPENDIX C. TIME COMPLEXITIES CASE BY CASE

From the cost of each algorithm, the total cost of the generalization method can be derived by considering the possible user actions relevant to generalization and which algorithms they trigger:

—*New formula/cell.* A new formula is entered or brought onto the screen, or an existing formula is modified. There are two possible subcases: either the new formula caused a possible cycle or it did not.

—*Several new formulas/cells.* Several existing cells are moved onto the screen, such as when an entire form is displayed. This is the same case as the previous one except for its volume.

—*A form is saved or chosen for removal from the screen or memory.* At least all the cells on the form must be generalized; in some cases all the cells on the screen must be generalized, as was discussed earlier.

—*A new instance is created by copying from a model.*

Table C-I recapitulates the costs presented in the main paper of each algorithm, and Table C-II uses these costs to show the time complexity for each of the above user actions. As Table C-II shows, the worst-case costs are in every case based on the size of the on-screen portion of the program as opposed to the size of the entire program. We have mentioned that in the Forms/3 implementation of the method, some cells that are not actually visible are classified as "on-screen," such as cells on obscured or in iconified windows. However, the

Table C-I. Summary of Algorithm Costs

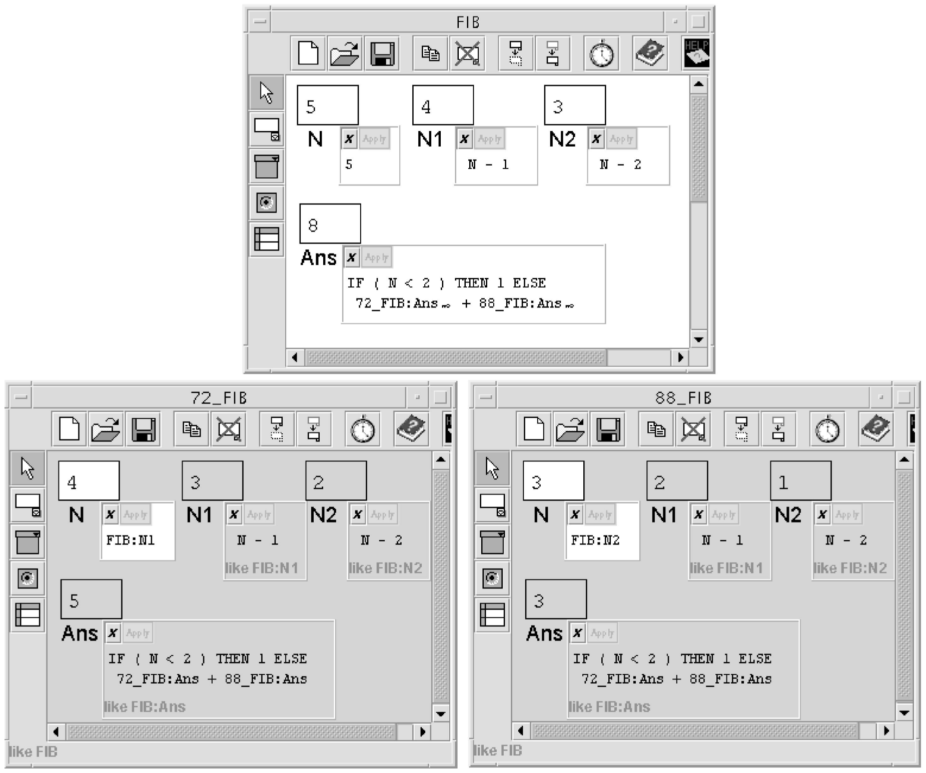
Algorithm	Description	Time Complexity
`Insert(le)`	Add labeled edge `le`.	O(|V|)
`Delete(le)`	Delete labeled edge `le`.	O(|V|)
`Find(le)`	Find labeled edge `le`.	O(|V|)
`Cycle?(aCell)`	Detect possible cycles in subgraph rooted at cell `aCell`.	O(|E|)
`RemoveICE`	Remove ICE edges.	O(|E|)
`Generalize`	Record generalized formula relationships (cost shown is if triggered by editing one cell's formula).	O(|V|*|E|)

Table C-II. Costs by User Action

User Action	Algorithms Invoked (from Table C-I)	Time Complexity
New formula/cell on the screen, possible cycle	`Insert,Delete,Find`: called for each reference in (constant-length) formula.	O(|V|)
	`Cycle?`: called once.	O(|E|)
	`RemoveICE`: called once.	O(|E|)
	`Generalize`: Called on one cell.	O(|V|*|E|)
	Total	O(|V|*|E|)
New formula/cell on the screen, no cycle	`Insert,Delete,Find`: called for each reference in (constant-length) formula.	O(|V|)
	`Cycle?`: called once.	O(|E|)
	Total	O(|V| + |E|)
Several new formulas/cells on the screen.	Same as new formula (above), but repeat for each cell.	O(|V|^2*|E|)
Any action requiring generalization of a single entire form	`RemoveICE`: called once.	O(|E|)
	`Generalize`: Called on multiple cells.	O(|V|^2*|E|)
	`Delete,Find`: In cases where cells are being removed from the screen, up to |E| edges may need to be removed.	O(|V|*|E|)
	Total	O(|V|^2*|E|)
Any action requiring generalization of the entire screen	Same as row above.	O(|V|^2*|E|)
New instance (copy)	Cost of generalization of single form.	O(|V|^2*|E|)
	`Insert,Find`: Add edges for up to |V| cell formulas.	O(|V|^2)
	Total	O(|V|^2*|E|)

method does not require this or any other particular definition of "on-screen" to succeed. Any reasonable definition will suffice, because taking a cell out of this status is one of the events triggering generalization. For example, to precisely bound generalization cost by the number of pixels on the screen, a variation that we have considered is for the definition of "on-screen" to exclude obscured cells and cells on iconified windows. Under such a change, whenever the user iconified a form or obscured a cell, generalization would be triggered (for the affected cells). This change would generate more calls to the generalization algorithm than presently occur, but each call would potentially process a smaller SSCG, since fewer cells would be considered to be on-screen. However, there has been no reason to make this change, as generalization time in our implementation is already fast enough to not introduce noticeable delays, as was demonstrated in Section 6.

APPENDIX D. ADDITIONAL PROGRAMS

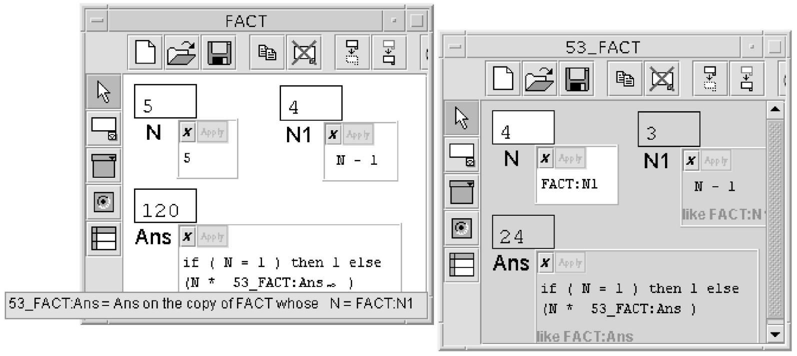


```
FIB
5  N    5
4  N1   N - 1
3  N2   N - 2
8  Ans  IF ( N < 2 ) THEN 1 ELSE
         72_FIB:Ans∞ + 88_FIB:Ans∞

72_FIB
4  N    FIB:N1
3  N1   N - 1            like FIB:N1
2  N2   N - 2            like FIB:N2
5  Ans  IF ( N < 2 ) THEN 1 ELSE
         72_FIB:Ans + 88_FIB:Ans
        like FIB:Ans

88_FIB
3  N    FIB:N2
2  N1   N - 1            like FIB:N1
1  N2   N - 2            like FIB:N2
3  Ans  IF ( N < 2 ) THEN 1 ELSE
         72_FIB:Ans + 88_FIB:Ans
        like FIB:Ans
```

Fig. D-1. A version of Fibonacci as it might be programmed by a more traditional programmer than the one who created Figure 4. This version is hierarchical—its relationships are those of a recursive "call tree"—which makes it much more traditional than Figure 4's. However, it still requires generalization for the same reasons as for Figure 4.



```
FACT
5    N    5
4    N1   N - 1
120  Ans  if ( N = 1 ) then 1 else
          (N *  53_FACT:Ans∞ )
53_FACT:Ans = Ans on the copy of FACT whose  N = FACT:N1

53_FACT
4    N    FACT:N1
3    N1   N - 1     like FACT:N
24   Ans  if ( N = 1 ) then 1 else
          (N *  53_FACT:Ans )
          like FACT:Ans
```

Fig. D-2. The factorial program from Section 6. The user has used the key icon to view the generalized meaning (at bottom left) of the "sample" reference to 53_Fact:Ans.

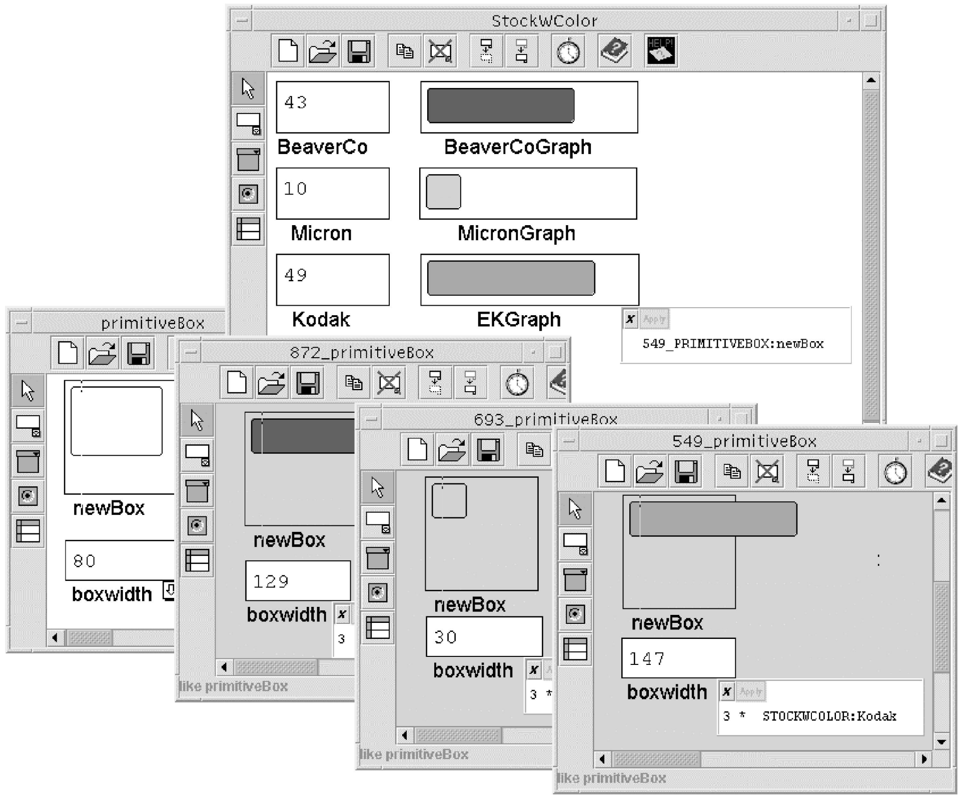


Fig. D-3. The stocks example program from Section 6. This is an investment visualization application. The rightmost cells in the `StockwColor` window form a horizontal color bar graph comparing different stock prices.